Learning Motion Manifolds with Convolutional Autoencoders

Learning Motion Manifolds with Convolutional Autoencoders

Daniel Holden[1], Jun Saito[2], Taku Komura[1], Thomas Joyce[1]
[1]University of Edinburgh, [2]Marza Animation Planet

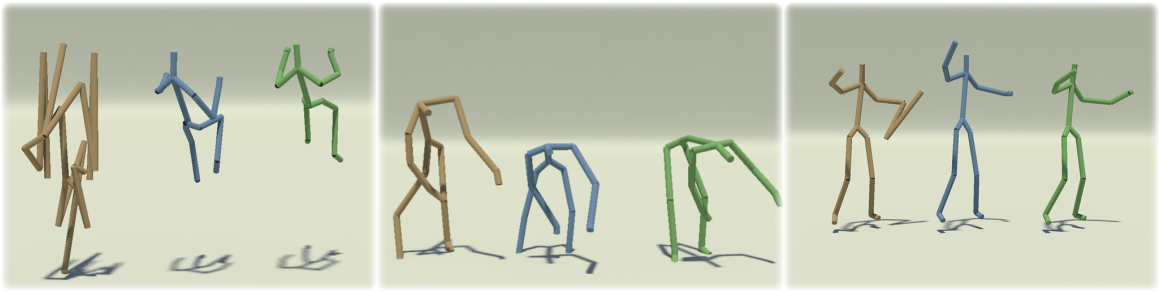

Figure 1: *A set of corrupted motions (orange) fixed by projection onto the motion manifold (blue) and compared to the ground truth (green). Motion corrupted with artificial noise that randomly sets values to zero with a probability of 0.5 (left), motion data from Kinect motion capture system (center), and motion data with missing the wrist marker (right).*

Abstract

We present a technique for learning a manifold of human motion data using Convolutional Autoencoders. Our approach is capable of learning a manifold on the complete CMU database of human motion. This manifold can be treated as a prior probability distribution over human motion data, which has many applications in animation research, including projecting invalid or corrupt motion onto the manifold for removing error, computing similarity between motions using geodesic distance along the manifold, and interpolation of motion along the manifold for avoiding blending artefacts.

Keywords: deep neural networks, convolutional neural networks, autoencoding, machine learning, motion data, animation, character animation, manifold learning

1 Introduction

Motion data is typically represented as a time-series where each frame represents some pose of a character. Poses of a character are usually parametrized by the character joint angles, or joint positions. This representation is excellent for data processing, but valid human motion only exists in a small subspace of this representation. It is easy to find a configuration in this space which does not represent valid human motion - either by setting the joint angles to extreme configurations and making poses that are biologically impossible, or by creating large differences between frames of the time-series such that the motion is too fast for humans to achieve.

It is of great interest to researchers to find good techniques for finding this subspace of valid motion, called the motion manifold. This is because it allows for data processing tasks to be defined with respect to it - such as interpolating motion only through the space of valid motions - or taking distances which do not pass through the space of impossible motion.

Our contribution is the following:

- Unsupervised learning of a human motion manifold by a Convolutional Autoencoder
- Application of our manifold in various areas of animation research.

2 Related Work

In this section we will briefly discuss works aimed at learning a manifold over the space of motion data and then works related to Convolutional Neural Networks.

Learning the motion manifold It is of interest to researchers in machine learning and computer graphics to produce motion manifolds from human motion data. PCA is an effective approach for modelling the manifold of human gait, although it cannot handle many different types of motion altogether. Chai et al. [2005] apply local PCA to produce a motion manifold that includes of various types of human motion data, and apply it for synthesizing movements from low dimensional signals. Lee et al. [2010] propose a data structure called motion fields where the user can interactively control the characters in the motion manifold. These methods require a lot of preprocessing, including computing the distance between frames and producing structures such as KD trees for handling large amounts of data. Rose et al. map the hand position to the full body motion using Radial Basis Functions (RBF). Mukai and Kuriyama [2005] apply Gaussian Processes (GP) to produce a smoother mapping. Lawrence [2003] proposes to apply Gaussian Process Latent Variable Model (GPLVM) to produce a mapping between low dimensional latent space and high dimensional data. Grochow et al. [2004] apply GPLVM to human motion synthesis. Wang et al. [2005] apply GPLVM to develop dynamic models of human motion. Non-linear kernel based methods such as RBF/GP cannot scale as the covariance matrix grows as the square of the number of data points used. Additionally these works suffer from the same issue as other works of only encoding the temporal aspect of motion once a manifold has been built for human poses.

Neural networks and its application to motion data Convolutional Neural Networks (CNNs) have been used effectively in many machine learning tasks. They have achieved particular success in the domain of image classification [Krizhevsky et al. 2012; Ciresan et al. 2012] but they have additionally been used for many other tasks including video classification[Karpathy et al. 2014], facial recognition [Nasse et al. 2009], action recognition [Ji et al. 2013], tracking [Fan et al. 2010] and speech recognition [Abdel-Hamid et al. 2014; Ji et al. 2013]. Our work applies these techniques to the domain of learning a motion manifold on human motion data.

Taylor et al. [2011] use the notion of a Conditional Restricted Boltzmann machine to learn a time-series predictor which can predict the next pose of a motion given several previous frames. While this approach does not rely on clustering or processing of pose data its capacity is still limited because with lots of data there exists large ambiguity in generation of the next pose which can lead to either the motion "dying out" and reaching the average pose, or high frequency noise introduced by sampling.

The difference in our approach is to use convolution to explicitly encode the temporal aspect of motion with equal importance to the pose subspace.

3 Notations

Given a time-series of human poses $\mathbf{X} \in \mathbb{R}^{nm}$ where n is the number of frames and m is the number of degrees of freedom in the representation of the pose, we wish to learn some manifold projection operator $\mathbf{Y} = \mathbf{\Phi}(\mathbf{X})$, $\mathbf{Y} \in [-1, 1]^{ik}$ and some inverse projection operator $\hat{\mathbf{X}} = \mathbf{\Phi}^{\dagger}(\mathbf{Y})$, $\hat{\mathbf{X}} \in \mathbb{R}^{nm}$ such that the application of the inverse of the projection operation $\mathbf{\Phi}^{\dagger}$ always produces a $\hat{\mathbf{X}}$ which lies within the subspace of valid human motion. Here i, k are the number of frames and degrees of freedom on the motion manifold, respectively. The space $[-1, 1]^{ik}$ therefore represents the parametrisation of the manifold. As this space is bounded by -1 and 1, so is the manifold, and in areas outside this bound the inverse projection operation is undefined. Additionally, the uniform distribution of values of $\hat{\mathbf{X}}$ drawn from $[-1, 1]^{ik}$ represents a prior probability distribution over the space of human motion which can be augmented with other beliefs for use in machine learning tasks.

In this paper we learn the projection operation $\mathbf{\Phi}$ and inverse projection operation $\mathbf{\Phi}^{\dagger}$ using a Deep Convolutional Autoencoder [Vincent et al. 2010]. The *Visible Units* correspond to $\mathbf{X}$ and the values of the *Hidden Units* at the deepest layer corresponds to $\mathbf{Y}$. Propagating the input through the network is therefore the projection operator $\mathbf{\Phi}$, and propagating the *Hidden Units* values backward to reconstruct *Visible Units* is the inverse operation $\mathbf{\Phi}^{\dagger}$.

4 Data Preprocessing

In this section we explain our construction of values of $\mathbf{X}$ using time-series data from the CMU motion capture database.

We use the full CMU Motion Capture Database [Cmu] which consists of 2605 recordings of roughly 10 hours of human motion, captured with an optical motion capture system. We sub-sample this data to 30 frames per second and separate it out into overlapping windows of 160 frames (overlapped by 80 frames). This results in 9673 windows. We choose 160 frames as it roughly covers the period of most distinct human actions (around 5 seconds) - but in our framework this dimension is variable in size.

We normalize the joint lengths, and use the joint positions of 20 of the most important joints. Global translation around the XZ plane is removed, and global rotation around the Y axis removed.

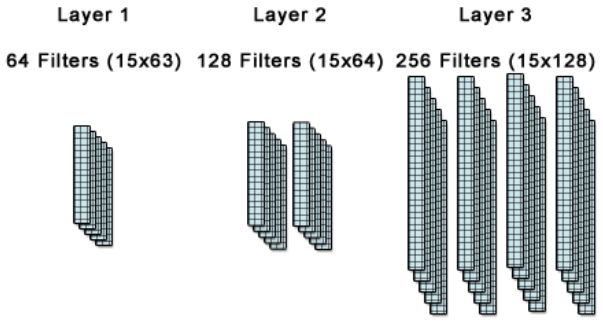

Figure 2: *Structure of the Convolutional Autoencoder. Layer 1 contains 64 filters of size 15x63. Layer 2 contains 128 filters of size 15x64. Layer 3 contains 256 filters of size 15x128. The first dimension of the filter corresponds to a temporal window, while the second dimension corresponds to the number of features/filters on the layer below.*

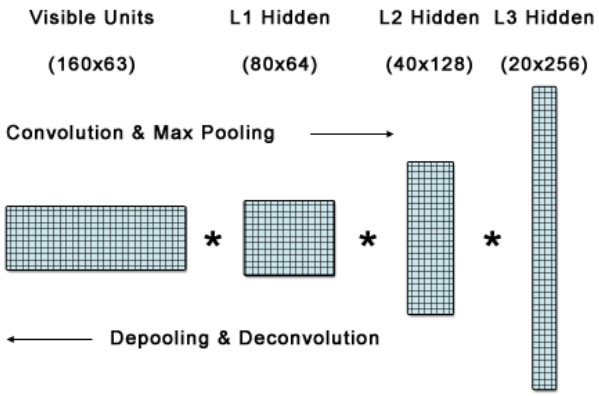

Figure 3: *Units of the Convolutional Autoencoder. The input to layer 1 is a window of 160 frames of 63 degrees of freedom. After the first convolution and max pooling this becomes a window of 80 with 64 degrees of freedom. After layer 2 it becomes 40 by 128, and after layer 3 it becomes 20 by 256.*

The rotational velocity around the Y axis and the translational velocity in the XZ plane relative to the character's forward direction are included in the input vector. These can be integrated over time to recover the global position and rotation of the character in an animation. Finally we subtract the mean pose from each pose vector before input to the network. This results in a final input vector with a window size of 160 and 63 degrees of freedom. $\mathbf{X} \in \mathbb{R}^{160 \times 63}$.

5 Convolutional Neural Networks for Learning Motion Data

In this section we will explain the structure of the Convolutional Autoencoder. Readers are referred to tutorials such [DeepLearning] for the basics of Convolutional Neural Networks. We construct and train a three-layer Convolutional Autoencoder. An overview of our network structure is given by Fig. 2 and an overview of the transformations is given by Fig. 3.

Each layer of the CNN performs a one-dimension convolution over the temporal domain independently for a number of filters. The output of the convolution is called a *feature map*, and represents the presence of that particular filter at a certain point in time. A filter

size of 15 is chosen as it roughly corresponds to small actions on the lower layers (half a second), and larger actions on the deeper layers (several seconds). Filter values $\mathbf{W}$ are initialised to some small random values found using the "fan-in" and "fan-out" criteria [Hinton 2012], while biases $\mathbf{b}$ are initialised to zero.

Now we describe about the forward process in each layer. First the input data is convolved, and then a one-dimensional max pooling step is used to shrink the temporal domain - taking the maximum of each consecutive pair of frames. Max pooling provides a level of temporal invariance and encourages filters to learn separate distinct features of the input. Finally the output is put through the $\tanh$ function. This places it in the range $[-1, 1]$, which compresses the input, effectively clipping it to the subspace, as well as allowing the neural network to construct a non-linear manifold.

For a single layer k, given the convolution operator $*$, max pooling operator $\mathbf{\Psi}$, filter weights $\mathbf{W}_k$ and biases $\mathbf{b}_k$, the projection operation is given by the following.

$$\mathbf{\Phi}_k(\mathbf{X}) = \tanh(\mathbf{\Psi}(\mathbf{X} * \mathbf{W}_k + \mathbf{b}_k)) \tag{1}$$

Now we describe about the backward process in each layer. This is done by simply inverting each component of the network, taking special care to invert the pooling operator $\mathbf{\Psi}$. In general this operator is non-invertible, so instead we formulate two approximate inverses that can be used in different cases. When training the network, which we describe shortly in Section 6, a single individual pooling location is picked randomly and the pooled value is placed there, setting the other pooling location to zero. This is a good approximation for the inverse of max pooling process, although it results in some noise in the produced animation. When training in the fine tuning stage, which is the second stage of the training stage that we describe in Section 6, or when performing projection to generate the animations shown in the results, the pooled value is instead distributed evenly across both pooling locations. This is not an accurate approximation of the inverse of the maximum operation, but it does result in smooth animation with no high frequency noise.

The inverse projection for some layer k is then given as the following. Where $\tilde{\mathbf{W}}_k$ represents the weights matrix flipped on the temporal axis and transposed on the axes representing each filter on this layer and the filters on the layer below. The $\mathbf{\Psi}^\dagger$ function represents the inverse pooling operation described above.

$$\mathbf{\Phi}_k^\dagger(\mathbf{Y}) = (\mathbf{\Psi}^\dagger(\tanh^{-1}(\mathbf{Y})) - \mathbf{b}_k) * \tilde{\mathbf{W}}_k \tag{2}$$

6 Training

We train the network using Denoising Autoencoding. The input $\mathbf{X}$ is corrupted to some new value $\mathbf{X}_c$ and the network is trained to reproduce the original input. The corrupted input $\mathbf{X}_c$ is produced by randomly setting values of the input $\mathbf{X}$ to zero with a probability of 0.1.

Training is first done individually for each layer. Once each layer is trained, the whole network is trained in a stage called *fine tuning*. Training is posed as an optimisation problem. We find weights $\mathbf{W}$ and biases $\mathbf{b}$ such that the following loss function is minimised.

$$Loss(\mathbf{X}) = \|\mathbf{X} - \mathbf{\Phi}^\dagger(\mathbf{\Phi}(\mathbf{X}_c))\|_2^2 + \alpha \|\mathbf{\Phi}(\mathbf{X}_c)\|_1 \tag{3}$$

Where $\|\mathbf{X} - \mathbf{\Phi}^\dagger(\mathbf{\Phi}(\mathbf{X}_c))\|_2^2$ measures the squared reproduction error and $\|\mathbf{\Phi}(\mathbf{X}_c)\|_1$ represents an additional sparsity constraint that ensures the minimum number of hidden units are active at a time, so that independent local features are learned. This is scaled by some small constant α which in our case is set to 0.01.

We make use of automatic derivatives calculated by Theano [Bergstra et al. 2010] and iteratively input each data point while updating the filters in the direction that minimises the loss function. We use some learning rate initially set to the value of 0.5 and slowly decayed by a factor of 0.9 at each epoch. Each layer is trained for 25 epochs.

In the fine tuning stage a different learning rate of 0.01 is used and the constant α is set to zero. Additionally, as mentioned in Section 5, the alternative method of max pooling is used, where values are distributed evenly.

Once training is complete the filters express strong temporal and inter-joint correspondences. Each filter expresses the movement of several joints over a period of time. Each of these filters therefore should correspond to natural, independent components of motion. Filters on deeper layers express the presence of filters on lower layers. They therefore represent higher level combinations of the different components of motion.

7 Results

In this section we demonstrate some of the results of our method, including fixing corrupted motion data, filling in missing motion data, interpolation of two motions, and distance comparisons between motions. The readers are referred to the supplementary video for additional results.

Fixing Corrupt Motion Data Projection and inverse projection can be used to fix corrupted motion data. In Fig. 1, left, we artificially corrupt motion data by randomly setting values to zero with a probability of 0.5, removing half of the information. After projection onto the manifold the motion data is effectively recovered and matches closely to the ground truth.

We record motion data by simultaneously capturing motion with the Microsoft Kinect and a separate inertia based motion capture system. The Kinect capture contains many errors, but after projection onto the motion manifold it far more closely matches the ground truth data found from the inertia based motion capture system (see Fig. 1, center).

Filling in Missing Data Projection can be used to fill in missing motion data. We use our technique to automatically infer the marker position of a character's wrist (see Fig. 1, right). We can also produce some stepped animation data using only every 15 frames and project this onto the manifold. Our method inserts smooth transitions between the stepped keyframes, corresponding to motion found in the training data.

Motion Interpolation We interpolate two distinctly separate motions of *doing press ups* and *walking forward*. Compared to a linear interpolation, which creates an impossible walk, interpolation along the manifold creates motion which remains valid.

Motion Comparison Using Naive Euclidean distance a motion of a confident walk with a turn is found to be similar to some odd motions, such as a sidestepping motion and an old man's walk. By taking the $\min$ and $\max$ over the temporal dimension i of the Hidden Units and consequently taking Euclidean distance, we get a distance measure G which is temporally invariant and correctly matches high level features. It only finds motions with confident walks and turns to be similar.

$$G(\mathbf{X}_0, \mathbf{X}_1) = \| \max_i(\mathbf{\Phi}(\mathbf{X}_0)) - \max_i(\mathbf{\Phi}(\mathbf{X}_1))\|_2^2 + \| \min_i(\mathbf{\Phi}(\mathbf{X}_0)) - \min_i(\mathbf{\Phi}(\mathbf{X}_1))\|_2^2 \quad (4)$$

Computational Time Training is done using a Quadro K2000 graphics card with 2GB of memory. Training the full network takes 5 hours and 4 minutes. Where Layer 1 takes 43 Minutes, Layer 2 takes 51 Minutes, Layer 3 takes 1 hour and 17 Minutes and fine tuning takes 2 hours and 13 Minutes. At runtime the system only requires the multiplication of several medium sized matrices and so projection can be done in a fraction of a millisecond.

Comparison with Other Approaches Our method scales well compared to other approaches. Many methods that can handle large amounts of data require preprocessing that data using data structures such as KD trees. This is a very computationally expensive process, especially for high dimensional data. Our approach can handle a very large amount of data in a reasonable amount of time. The only methods that requires little data preprocessing and are scalable are classic methods such as PCA. Although PCA is good in many aspects, it does not perform well when non-linearity plays an important role such as when interpolating motions of different style or computing distances between motion data considering synchronization. The readers are referred to the supplementary video for visual comparison.

Limitations & Future Work Currently the manifold projection cannot guarantee constraints such as ground contact and joint lengths. In future we wish to explicitly incorporate these into our system using a constrained projection operator. We also wish to thoroughly evaluate our work against alternative methods and justify the effectiveness of using a deep architecture.

8 Conclusion

Learning a motion manifold has many useful applications in character animation. We present an approach appropriate for learning a manifold on the whole CMU motion capture database and present a number of operations that can be performed using the manifold.

The strength of our approach is in the explicit formulation of the temporal domain of the manifold. The position of a single joint is not just linked to the position of other joints in the same pose - but to the position of other joints in separate frames of the animation. This makes our system more powerful and easier to use than approaches, which first attempt to first learn a manifold over the pose space and then augment it with a statistical model of transitions.

References

ABDEL-HAMID, O., MOHAMED, A.-R., JIANG, H., DENG, L., PENN, G., AND YU, D. 2014. Convolutional neural networks for speech recognition. *IEEE/ACM Trans. Audio, Speech and Lang. Proc. 22*, 10 (Oct.), 1533–1545.

BERGSTRA, J., BREULEUX, O., BASTIEN, F., LAMBLIN, P., PASCANU, R., DESJARDINS, G., TURIAN, J., WARDE-FARLEY, D., AND BENGIO, Y. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

CHAI, J., AND HODGINS, J. K. 2005. Performance animation from low-dimensional control signals. *ACM Trans. on Graph. 24*, 3.

CIRESAN, D., MEIER, U., AND SCHMIDHUBER, J. 2012. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 3642–3649.

CMU. Carnegie-Mellon Mocap Database. `http://mocap.cs.cmu.edu/`.

DEEPLEARNING. Deep learning convolutional neural networks. `http://deeplearning.net/tutorial/lenet.html`.

FAN, J., XU, W., WU, Y., AND GONG, Y. 2010. Human tracking using convolutional neural networks. *Neural Networks, IEEE Transactions on 21*, 10 (Oct), 1610–1623.

GROCHOW, K., MARTIN, S. L., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. In *ACM Transactions on Graphics (TOG)*, vol. 23, ACM, 522–531.

HINTON, G. 2012. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade*, G. Montavon, G. Orr, and K.-R. Mller, Eds., vol. 7700 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 599–619.

JI, S., XU, W., YANG, M., AND YU, K. 2013. 3d convolutional neural networks for human action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 35*, 1 (Jan), 221–231.

KARPATHY, A., TODERICI, G., SHETTY, S., LEUNG, T., SUKTHANKAR, R., AND FEI-FEI, L. 2014. Large-scale video classification with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 1725–1732.

KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 1097–1105.

LAWRENCE, N. D. 2003. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in Neural Information Processing Systems*, None.

LEE, Y., WAMPLER, K., BERNSTEIN, G., POPOVIĆ, J., AND POPOVIĆ, Z. 2010. Motion fields for interactive character locomotion. In *ACM Trans. Graph.*, vol. 29, ACM, 138.

MUKAI, T., AND KURIYAMA, S. 2005. Geostatistical motion interpolation. *ACM Trans. Graph. 24*, 3, 1062–1070.

NASSE, F., THURAU, C., AND FINK, G. 2009. Face detection using gpu-based convolutional neural networks. In *Computer Analysis of Images and Patterns*, vol. 5702 of *Lecture Notes in Computer Science*. 83–90.

TAYLOR, G. W., HINTON, G. E., AND ROWEIS, S. T., 2011. Two distributed-state models for generating high-dimensional time series.

VINCENT, P., LAROCHELLE, H., LAJOIE, I., BENGIO, Y., AND MANZAGOL, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research 11*, 3371–3408.

WANG, J., HERTZMANN, A., AND BLEI, D. M. 2005. Gaussian process dynamical models. In *Advances in neural information processing systems*, 1441–1448.